# LEARNING FROM OBSERVATIONS

**Outline**

- Forms of Learning
- Inductive Learning
- Learning Decision Trees
- Ensemble Learning

Based on the textbook by Stuart Russell & Peter Norvig:

*Artificial Intelligence, A Modern Approach (2nd Edition)*

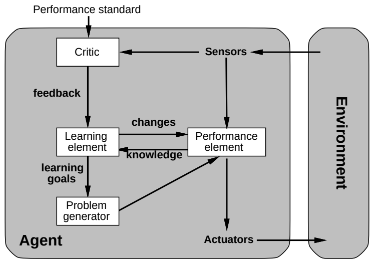
Chapter 18; excluding Section 18.5





# FORMS OF LEARNING

Recall the design of a **learning agent** from Chapter 2:




1. A *performance element* (a conventional agent) is responsible for choosing external actions.
2. A *learning element* aims to improve the agent.
3. A *critic* evaluates the performance of the agent.
4. A *problem generator* suggests new courses of action.





## Designing a Learning Element

The design of the learning element if affected by four major factors:

1. Which *components* of the performance element are to be learned.
2. What *feedback* is available to learn these components.
3. What *representation* is used for the components.
4. The availability of *prior knowledge* on what is being learned.

**Examples.** Even newborn babies exhibit knowledge of the world.

Consider a physicist vs. art critic examining a stack of bubble chamber photographs.





## Which Components Can Be Learned?

- The following components of agents can be learned:
  1. A direct mapping from the current state to actions.
  2. A means to infer relevant properties of the world from the percept sequence.
  3. Information about the way the world evolves.
  4. Information about the possible outcomes of the agent's actions.
  5. Utility information indicating the desirability of world states.
  6. Goals describing states that maximize the agent's utility.
- Various kinds of internal representations can be used for the components: polynomials, logical rules, Bayesian networks, etc.

## Available Feedback

The field of *machine learning* usually distinguishes three cases:

- **Supervised learning** involves learning a *function* from examples of its inputs and outputs (provided by an external *teacher*).
- In **unsupervised learning**, the correct outputs are not known, but one may learn patterns in the input.

  **Example.** An unsupervised learner may learn to predict its future percepts given its percept history so far.
- **Reinforcement learning**: the outputs get evaluated somehow (for instance, the agent receives a *reward* or a *punishment*), but the correct outputs remain unknown.

Any *prior knowledge* on the environment helps enormously in learning!





## INDUCTIVE LEARNING

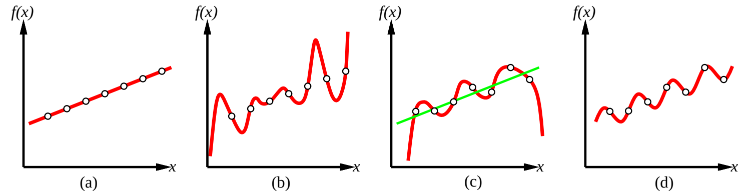
- In general, learning can be understood as a process of determining a representation for some function $f$ of interest.
- An **example** is a pair $\langle x, f(x)\rangle$ where $x$ is the input and $f(x)$ is the output of the function $f$ applied to $x$.
- The task of **pure inductive inference** (or **induction**) is:

  *Given a collection of examples of $f$, return a function $h$ (called a **hypothesis**) that approximates $f$.*
- There are often many hypotheses conforming to the examples and it is hard to tell whether any particular $h$ is a good approximation.
- A good hypothesis $h$ will **generalize** well, i.e. predict unseen examples correctly.





**Example.** Consider a set of points $\langle x,y\rangle$ in $xy$-plane such that $y = f(x)$. The task is to find $h(x)$ that fits the points well.




- As $f$ is unknown, there are many choices for $h$.
- In Figures (a)–(b), the set of polynomials (of degree at most $k$) is used as the **hypothesis space**.
- A **consistent** hypothesis agrees with all examples.





## Choosing a Consistent Hypothesis

- One principle is **Ockham's razor**: prefer the *simplest* hypothesis consistent with the data in order to extract a *pattern* from it.
- Figure (c) shows another set of examples which is difficult to capture using polynomials (a degree-6 polynomial is required).
- In this case, a linear approximation is able to predict the data better than polynomials of higher degree.
- Figure (d) shows how an exact fit is obtained with a simple function of the form $ax + b + c\sin x$.
- The (im)possibility of finding a simple, consistent hypothesis depends strongly on the hypothesis space chosen.

## Choosing the Hypothesis Space

➤ A learning problem is **realizable** if the hypothesis space contains the true function and **unrealizable** otherwise.

➤ Sometimes *prior knowledge* can help to derive a hypothesis space in which the true function is known to exist.

➤ The use of unnecessarily large hypothesis spaces (e.g. Turing machines) is ruled out by the *complexity* of learning:

> *"There is a trade-off between expressiveness of a hypothesis space and the computational complexity of finding simple, consistent hypothesis within that space."*

➤ Another reason to prefer simpler hypothesis spaces is that the resulting representations may be more efficient to use.





## LEARNING DECISION TREES

➤ A **decision tree** is a representation of a function $f$ from an $n$-dimensional attribute space to the set $\{Yes, No\}$.

Thus $f$ can be understood as a Boolean-*valued* function.

➤ Decision trees are structured as follows:

1. Each internal node tests the value of an attribute and the branches are labeledby the values of the attribute.
2. Leaf nodes contain the $Yes/No$ answer for the **goal predicate** the values of which are represented by the decision tree.

➤ Arbitrary Boolean functions can be represented as decision trees.

➤ The Non-Boolean case with more than two values can be covered.





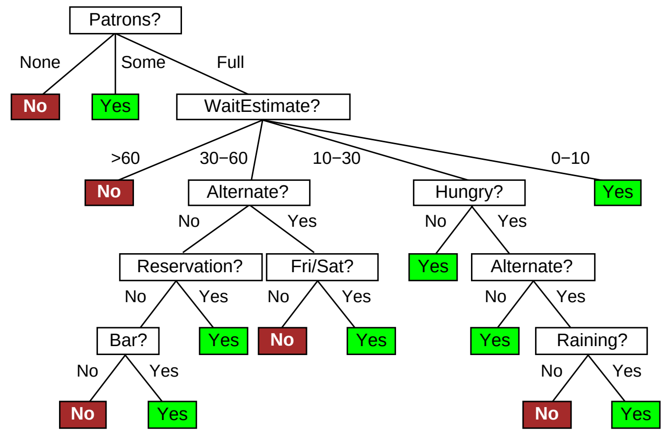
**Example.** Consider the problem of deciding whether to wait for a table at a restaurant. The aim is to learn a decision tree for the goal predicate *WillWait* using the following attributes:

1. *Alternate*: is there a suitable alternative restaurant nearby?
2. *Bar*: is there a comfortable bar area to wait in?
3. *Fri/Sat*: is it Friday or Saturday?
4. *Hungry*: are we hungry?
5. *Patrons*: the number of people (*None*, *Some*, *Full*) in the restaurant.
6. *Price*: the price range of the restaurant ($, $$, $$$).
7. *Raining*: is it raining outside?
8. *Reservation*: have we made a reservation beforehand?
9. *Type*: the type (*French*, *Italian*, *Thai*, *Burger*) of the restaurant.
10. *WaitEstimate*: the estimate in minutes (*0-10*, *10-30*, *30-60*, *>60*).





**Example.** Mr. Russell makes decisions for this domain as follows:




☞ *Price* and *Type* attributes are considered irrelevant.

## Expressiveness of Decision Trees

- A decision tree hypothesis for *WillWait* is an assertion of the form

$$\forall r(WillWait(r) \leftrightarrow P_1(r) \vee P_2(r) \vee \ldots \vee P_n(r))$$

where each $P_i(r)$ is a conjunction of tests corresponding to a path from the root to a leaf with a positive outcome.

- This makes decision trees effectively propositional and full first order logic is not easily covered.
- Any boolean function can be encoded as a decision tree, but such a representation may require a space exponential in the number of attributes, as for **parity** and **majority functions**.
- For $n$ Boolean attributes, there are $2^{2^n}$ different Boolean-valued functions. When $n = 6$, this number is about $1.8 \times 10^{19}$.





## Inducing Decision Trees from Examples

- An **example** is described by a combination of values for the attributes and the corresponding value of the goal predicate.
- The task is to form a decision tree for the predicate *WillWait* using a set of **positive** and **negative** examples as the **training set**:

| Example | Attributes | | | | | | | | | | Goal |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Alt* | *Bar* | *Fri* | *Hun* | *Pat* | *Price* | *Rain* | *Res* | *Type* | *Est* | *WillWait* |
| $X_1$ | Yes | No | No | Yes | Some | $$$ | No | Yes | French | 0–10 | Yes |
| $X_2$ | Yes | No | No | Yes | Full | $ | No | No | Thai | 30–60 | No |
| $X_3$ | No | Yes | No | No | Some | $ | No | No | Burger | 0–10 | Yes |
| $X_4$ | Yes | No | Yes | Yes | Full | $ | No | No | Thai | 10–30 | Yes |
| $X_5$ | Yes | No | Yes | No | Full | $$$ | No | Yes | French | >60 | No |
| $X_6$ | No | Yes | No | Yes | Some | $$ | Yes | Yes | Italian | 0–10 | Yes |
| $X_7$ | No | Yes | No | No | None | $ | Yes | No | Burger | 0–10 | No |
| $X_8$ | No | No | No | Yes | Some | $$ | Yes | Yes | Thai | 0–10 | Yes |
| $X_9$ | No | Yes | Yes | No | Full | $ | Yes | No | Burger | >60 | No |
| $X_{10}$ | Yes | Yes | Yes | Yes | Full | $$$ | No | Yes | Italian | 10–30 | No |
| $X_{11}$ | No | No | No | No | None | $ | No | No | Thai | 0–10 | No |
| $X_{12}$ | Yes | Yes | Yes | Yes | Full | $ | No | No | Burger | 30–60 | Yes |





## How to Construct a Decision Tree?

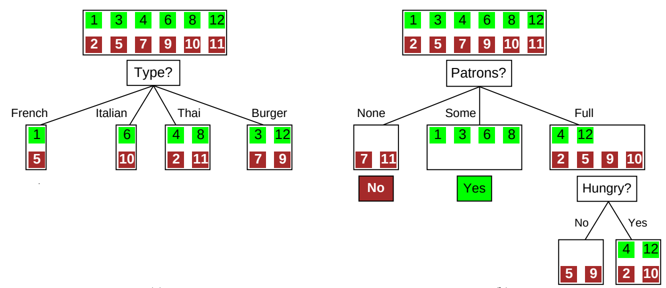
- A trivial solution encodes each example as a path leading to a leaf:
  1. Along the path, all the attributes are tested in turn.
  2. The leaf node holds the correct classification for the example.
- Such a decision tree produces correct classifications for the examples in the training set, but cannot extrapolate to others.
- By applying the *Ockham's razor* principle, we should find a "smallish" decision tree that is consistent with examples.
- Unfortunately, it is *intractable* to find the smallest decision tree for a training set, but relatively small ones can be found using a suitable heuristics.





- The basic idea is to test *the most important attribute first*, i.e., the one that best classifies examples in the training set.

**Example.** In the restaurant example, the attribute *Patrons* yields a much better classification than the attribute *Type*.




(a) (b)

## An Algorithm for Learning Decision Trees

➤ The process can be formalized as a concrete learning algorithm:

```
function DECISION-TREE-LEARNING(examples, attributes, default) returns a decision tree
  inputs: examples, set of examples
          attributes, set of attributes
          default, default value for the goal predicate

  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MAJORITY-VALUE(examples)
  else
     best ← CHOOSE-ATTRIBUTE(attributes, examples)
     tree ← a new decision tree with root test best
     for each value v_i of best do
        examples_i ← {elements of examples with best = v_i}
        subtree ← DECISION-TREE-LEARNING(examples_i, attributes − best,
                                          MAJORITY-VALUE(examples))
        add a branch to tree with label v_i and subtree subtree
     end
     return tree
```





➤ The training set is split into smaller sets of examples that are solved as recursive instances of the decision tree learning problem.

➤ The recursive problems fall into four different categories:

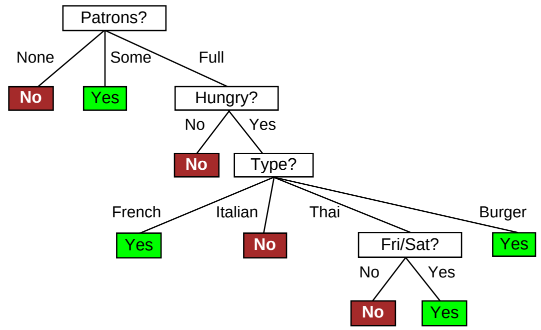
1. If there are both positive and negative example, then one of the best attributes is chosen to split the examples.
2. If all the remaining examples are positive (or all negative), then the answer is *Yes* (or *No*).
3. If there are no examples left, the majority classification at the node's parent is returned as a default value.
4. If there are no attributes left, but both positive and negative example, there is **noise** in the data *or* the set of attributes is insufficient to fully determine the goal predicate.

➤ A way to handle the last category is to use a majority vote.





**Example.** The following tree is obtained for the earlier training set:




➤ The resulting decision tree is much simpler than the original tree (which was actually used for generating the training set).

➤ Despite simplicity, the decision tree produces a correct classification for every example in the training set.





## Choosing Attribute Tests

➤ A perfect attribute divides the set of examples into subsets in which examples are all positive or all negative.

➤ One suitable measure for comparing attributes is the expected amount of **information** (in the sense proposed by Shannon) obtained by learning the exact values of attributes.

**Example.** Suppose you are going to bet 1€ on the flip of a coin.

1. If $P(Heads) = 0.99$, then $\mathrm{EMV} = 0.99 \times 1€ - 0.01 \times 1€ = 0.98€$ and $\mathrm{VPI}(Heads) = 1€ - 0.98€ = 0.02€$.
2. If $P(Heads) = 0.5$, then $\mathrm{EMV} = 0.5 \times 1€ - 0.5 \times 1€ = 0€$ and $\mathrm{VPI}(Heads) = 1€ - 0€ = 1€$.

☞ The less you know, the more valuable the information.

## Measuring Information Content

- Information theory uses the same intuition, but it measures information content in **bits** rather than value of information.
- In general one bit of information is enough to answer a yes/no question about which one has no idea.
- In general, the information content $I$ of the actual value of $V$ is

$$I(P(v_1), \ldots, P(v_n)) = \sum_{i=1}^{n} -P(v_i) \log_2 P(v_i) \text{ (bits)}.$$

where $P(v_1), \ldots, P(v_n)$ are the probabilities for the possible values $v_1, \ldots, v_n$ of the variable $V$.

**Example.** The information content $I(\frac{1}{2}, \frac{1}{2}) = -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{2}\log_2 \frac{1}{2} = 1$ bits, but $I(\frac{1}{100}, \frac{99}{100}) \approx 0.08$ bits.





## Information Gain

- In case of decision tres, the **information gain** from getting to know the exact value of a $v$-valued attribute $A$ is given by

$$\text{Gain}(A) = I(\frac{p}{p+n}, \frac{n}{p+n}) - \text{Remainder}(A)$$

where the *remaining information content*

$$\text{Remainder}(A) = \sum_{i=1}^{v} (\frac{p_i + n_i}{p+n} \times I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i}))$$

and $p$ ($p_i$) and $n$ ($n_i$) are the numbers of positive and negative examples (that have the $i^{\text{th}}$ value of $A$ in common).

**Example.** More information is gained from *Patrons* than from *Type*:

$$\text{Gain}(\textit{Patrons}) = 1 - [\tfrac{2}{12}I(0,1) + \tfrac{4}{12}I(1,0) + \tfrac{6}{12}I(\tfrac{2}{6}, \tfrac{4}{6})] \approx 0.541 \text{ and}$$

$$\text{Gain}(\textit{Type}) = 1 - [\tfrac{2}{12}I(\tfrac{1}{2}, \tfrac{1}{2}) + \tfrac{2}{12}I(\tfrac{1}{2}, \tfrac{1}{2}) + \tfrac{4}{12}I(\tfrac{2}{4}, \tfrac{2}{4}) + \tfrac{4}{12}I(\tfrac{2}{4}, \tfrac{2}{4})] = 0.$$





## Assessing the Performance of the Learning Algorithm

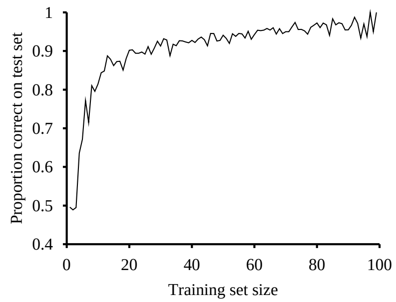
- A learning algorithm is good if it produces hypotheses which yield correct classifications for as many unseen examples as possible.
- A way to evaluate the performance of a learning algorithm is to
  1. Collect a large set of examples.
  2. Divide it into a **training set** and a separate **test set**.
  3. Apply the learning algorithm to the examples in the training set in order to generate a hypothesis $h$.
  4. Measure the percentage of examples in the test set that are correctly classifiedby the hypothesis $h$.
  5. Repeat steps 1 to 4 for random training sets of increasing size.





- The performance of a specific learning algorithm can be depicted as a **learning curve** that gives the percentage of correct classifications on the test set as a function of the training set size.
- To avoid **peeking** at test data, the selection of hypotheses should not be based on a fixed test set.

**Example.** The learning curve below shows how the decision tree learning algorithm performs in the restaurant example:



## Noise and Overfitting

- Recall the possibility of *noise* in the training set (there are two examples with identical attribute value, but classifications differ).
- **Overfitting** means that a (decision tree) learning algorithm forms a consistent hypothesis using *irrelevant attributes* for classification even when *relevant attributes* are missing.

**Example.** Consider the problem of predicting the roll of a die using

1. *Day*: the day on which the die was rolled,
2. *Month*: the month in which the die was rolled, and
3. *Color*: the color of the die.

A consistent (and totally spurious) hypothesis is found as long as no two examples have identical descriptions.





## How to Avoid Overfitting?

- The information gain is close to zero for irrelevant attributes.
- The relevance of attributes can also be tested using a statistical **significance test** based on known distributions.
- The total deviation

$$D = \sum_{i=1}^{v} \left( \frac{(p_i - \hat{p}_i)^2}{\hat{p}_i} + \frac{(n_i - \hat{n}_i)^2}{\hat{n}_i} \right)$$

  where $\hat{p}_i = p \times \frac{p_i + n_i}{p+n}$ and $\hat{n}_i = n \times \frac{p_i + n_i}{p+n}$ distributes according to the $\chi^2$ distribution with $v-1$ degrees of freedom.
- Decision trees can be **pruned** by neglecting irrelevant attributes.
- Pruning also helps to tolerate noise in the data.





## Broadening the Applicability of Decision Trees

To cover a wider variety of problems, many issues must be addressed.

1. **Missing data:** an example $X$ lacking the value of an attribute $A$ is given the majority classification among those obtained by assuming that $X$ has each value of $A$ in turn.
2. **Multivalued attributes:** when an attribute has a large number of possible values (e.g. *RestaurantName*), the information gain gives a misleading indication on the usefulness of the attribute.

   A solution is to use **gain ratio** instead of plain information gain.
3. **Continuous-valued attributes** (e.g. *Price*) are not well suited for decision-tree learning, and have to be **discretized** somehow.

   One technique is to decide **split points** using information gain.





## ENSEMBLE LEARNING

- The idea of **ensemble learning** is to select a collection (or **ensemble**) of hypotheses rather than a single hypothesis.
- A majority vote is used to combine the predictions of an ensemble.
- If each $h_i$ in the ensemble has a small error of $p$, then the probability of a misclassification becomes far more unlikely.

  **Example.** An ensemble of five hypotheses reduces an error rate of 1 in 10 down to an error rate of less than 1 in 100.
- The hypotheses chosen in the ensemble should be different in order to reduce the correlation between their errors.
- Ensemble learning provides a generic way of enhancing accuracy without increasing the complexity of the hypothesis space.

## Boosting

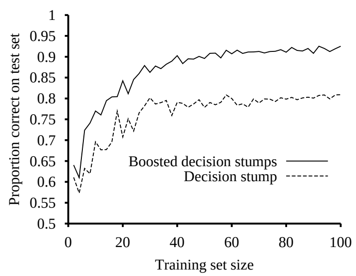
- A widely used ensemble method is **boosting** which is based on a **weighted training set** where initially $w_j = 1$ for every example $j$.
- At each round $0 < i \leq M$:
  1. a new hypothesis $h_i$ is generated;
  2. the weights of examples that are correctly/incorrectly classified under $h_i$ are decreased/increased.
- The final ensemble hypothesis is a weighted-majority combination of the hypotheses $h_1, \ldots, h_M$.
- A particular boosting algorithm (AdaBoost) has an attractive property: if applied to a **weak learning** algorithm, the resulting hypothesis classifies the training data perfectly for large enough $M$.

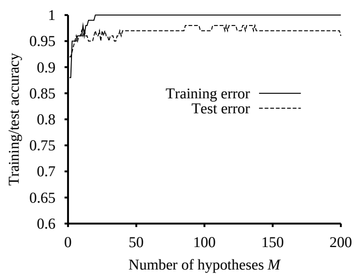




**Example.** Consider learning **decision stumps**, i.e. decision trees with a single test at the root, in the restaurant example.

- Unboosted decision stumps are not very effective for this data set.
- When boosting is applied (with $M = 5$) the performance is increased from 81% to 93% for 100 examples.
- The training set error reaches zero when $M$ is 20.








## SUMMARY

- Learning is essential for dealing with unknown environments.
- Learning may take several forms depending on the chosen representation, available feedback, and prior knowledge.
- The aim of **inductive learning** is to learn a **function** from examples of its inputs and outputs.
- **Ockham's razor** principle suggests choosing the simplest hypothesis that matches the examples observed.
- The performance of inductive learning algorithms is measured by their prediction accuracy as a function of the training set size.
- Ensemble methods such as **boosting** often perform better than individual methods.